TDT4137 ASSIGNMENT 4

Exercise 1

1 - a

Network i does not represent the environment because it represent the environment as if the wrapper and the shape influences the flavor, when in reality it is the opposite. But if we were to guess the flavor based on the wrapper and shape, this network could be used. Network ii have approximately the same issue, it suggest that the wrapper color influences the shape, which is not correct. Therefore the only correct bayesian network is iii, where both the wrapper and the shape is influenced by the flavor.

1 - b

Network iii is the correct network to represent the environment. This is because the flavor influences the shape and wrapper, which we have probabilities for. The other networks requires us to have probabilities like in network ii, where we would need to know how the wrapper influences shape (which it does not).

1 - c

Network i does in fact not assert that the wrapper and shape is independent. Two parental nodes sharing a child will be dependent on each other, this is because if we were given the flavor there would be a conditional relationship between the shape and the wrapper to infer that color.

1 - d

Result: 59% (see code)

1 - e

Result: 99.3% (see code)

1 - f

With s = 10 and a = 5: result: 8.5 (see code)

```python
p_flavor_strawberry = 0.7
p_flavor_anchovy = 0.3
p_roundshape_strawberry = 0.8
p_roundshape_anchovy = 0.1
p_redwrapper_strawberry = 0.8
p_redwrapper_anchovy = 0.1

def p_redwrapper():
    return p_flavor_strawberry * p_redwrapper_strawberry + p_flavor_anchovy * p_redwrapper_anchovy

def p_roundshape():
    return p_flavor_strawberry * p_roundshape_strawberry + p_flavor_anchovy * p_roundshape_anchovy

def p_red_and_round():
    return p_flavor_strawberry * p_redwrapper_strawberry * p_roundshape_strawberry + p_flavor_anchovy * p_redwrapper_anchovy * p_roundshape_anchovy

def p_strawberry_given_redwrapper_roundshape():
    # We want to calculate P(Strawberry | RedWrapper, RoundShape)
    # We can use Bayes' Theorem to calculate this
    # P(Strawberry | RedWrapper, RoundShape) = P(RedWrapper, RoundShape | Strawberry) * P(Strawberry) / P(RedWrapper, RoundShape)
    # P(RedWrapper, RoundShape | Strawberry) = P(RedWrapper | Strawberry) * P(RoundShape | Strawberry)
    return (p_redwrapper_strawberry * p_roundshape_strawberry * p_flavor_strawberry) / p_red_and_round()

def value_of_candybox (value_strawberry, value_anchovy):
    return p_flavor_strawberry * value_strawberry + p_flavor_anchovy * value_anchovy

print()
print(f"P(RedWrapper) = {p_redwrapper()}")
print(f"P(Strawberry | RedWrapper, RoundShape) = {p_strawberry_given_redwrapper_roundshape()}")
print(f"Value of CandyBox (v_strawberry: 20, value_anchovy: 5) = {value_of_candybox(10, 5)}")
```

Exercise 2

2 - a

Result: Mary chooses the guaranteed 500 dollars. (See code)

```python
def utility(x, R):
    return -(np.exp((-x)/R))

def a():
    print("\nPart A")
    R = 500
    guaranteed_return = 500
    risky_return = 5000
    chanche_risky = 0.6
    utility_guaranteed = utility(guaranteed_return, R)
    utility_risky = chanche_risky*utility(risky_return, R) + (1-chanche_risky)*utility(0, R)
    #Mary is rational and risk averse, so she will choose the highest expected utility
    # print(f"Utility of guaranteed return: {utility_guaranteed}")
    # print(f"Utility of risky return: {utility_risky}")
    if utility_guaranteed > utility_risky:
        print(f"Mary's choosen the guaranteed return of {guaranteed_return} with utility of {utility_guaranteed}\n")
        return guaranteed_return
    else:
        print(f"Mary's choosen the risky return of {risky_return} with utility of {utility_risky}\n")
        return risky_return
```

2 - b

Result: 152.399 (see code)

```python
def utility(x, R):
    return -(np.exp((-x)/R))

def b(initial_guess=100, epochs=1000, learning_rate=0.1):
    print("\nPart B")
    R = initial_guess
    guaranteed_return = 100
    risky_return = 500
    chanche_risky = 0.5
    for i in range(epochs):
        utility_guaranteed = utility(guaranteed_return, R)
        utility_risky = chanche_risky*utility(risky_return, R) + (1-chanche_risky)*utility(0, R)
        if utility_guaranteed > utility_risky:
            R += max(learning_rate, 0.1)
        else:
            R -= max(learning_rate, 0.1)

    print(f"R: {R}")
    print(f"Utility of guaranteed return: {utility_guaranteed}")
    print(f"Utility of risky return: {utility_risky}\n")
```